Parallel Global Routing for Standard Cells

Jonathan Rose
Computer Systems Laboratory
Center for Integrated Systems
Stanford University, Stanford CA 94305

N00014-87-K-0828

Abstract 1987

Standard cell placement algorithms have traditionally used cost functions that poorly predict the final area of the circuit, and so can result in placements with good wire length but large final area. A good estimation of the area can be obtained by *global routing* the placement, but routing has been considered too slow to be used as the placement metric. This paper presents a new, fast global routing algorithm for standard cells and its parallel implementation. The router is based on enumerating a subset of all two-bend routes between two points, and results in 16% to 37% fewer total number of tracks than the TimberWolf global router for standard cells [Sech85]. It is comparable in quality to a maze router and an industrial router, but is faster by a factor of 10 or more. Three *axes* of parallelism are implemented: wire-by-wire, segment-by-segment and route-by-route. Two of these approaches achieve significant speedup — route-by-route achieves up to 4.6 using eight processors, and wire-by-wire achieves from 10 to 14 using 15 processors. Because these axes are *orthogonal*, when combined we demonstrate that their respective speedups multiply each other. A simple model is used to predict speedups of up to 61 using 120 processors.

1 Introduction

The best way to evaluate a placement of circuit modules is to route it and determine the final area. Since routing is a time-consuming task typical placement algorithms [Hana72,Breu77] use other metrics such as total wire length or crossing counts that are easier to calculate. With the advent of usable commercial multiprocessors it is possible to consider using more compute-intensive cost functions *if* efficient parallel algorithms can be developed. The aim of the *Locus Project* is to integrate placement and routing into one optimization process, and to do this in a practical way, by using multiprocessing to increase the speed of the routing.

This paper presents the first step in the Locus Project: *LocusRoute*, a new global routing algorithm for standard cells, and its parallel implementation. Our goal is to make the recalculation time of an area-based cost function on a multiprocessor the same as conventional cost functions on a uniprocessor. The intention is for the global router to be invoked to rip-up and re-route wires whose end points have changed when one or more cells have been moved. This goal implies that routing time must be about one to two milliseconds per net on a VAX 11/780-class machine.

The routing performance of LocusRoute, as measured by total number of routing tracks, is better than that of TimberWolf 4.2 [Sech85] and is comparable to a maze router and an industrial router. It is fast because it investigates only a subset of two-bend routes between pairs of pins to be routed. The routing speed is increased further by parallelizing the algorithm in three ways: routing several wires at once, routing several two-point segments simultaneously, and evaluating possible two-bend routes in parallel. The wire-by-wire parallel approach achieves speedups ranging from 10 to 14 using 15 processors. The route-by-route approach achieves speedups of up to 4.6 using 8 processors. These two "axes" of parallelism are *orthogonal* to each other, and so when used in tandem their speedups will multiply. This is demonstrated on 15 processors, and used to predict speedups in excess of 60 using 120 processors for standard benchmark circuits.

Previous work on parallel routing [Breu81, Blan81, Adsh82, Nair82, Rute84, Iosu86, Won87] has generally focused on a fixed hardware mapping for the Lee routing algorithm [Lee61]. As such they lack the flexibility that is required in practical CAD software such as the global routers described in [Kamb85,Yama85]. Another drawback of special hardware for the Lee algorithm is that a uniprocessor implementation can be made very efficient using special software data structures that cannot be put easily into fixed hardware.

There have been few publications on global routing for standard cells, other than [Kamb85] and [Yama85] which give little detail of the process. The cost-model used in [Pate85] is similar to that used in LocusRoute. A survey of global routing that touches on standard cells appears in [Lore88]. An early version of this work was presented in [Rose88b].

This paper is organized as follows: Section 2 defines the global routing problem for standard cells and describes the serial LocusRoute algorithm. Section 3 gives performance comparisons with the Timberwolf 4.2 global router [Sech85], a maze router, and the UTMC Highland Router [Robe87]. Section 4 presents three approaches for speeding up the new router using parallel processing, and performance results. Section 5 presents experiments with combining two of the approaches which are then used to model and predict speedups for larger numbers of processors.

2 The LocusRoute Algorithm

This section defines the standard cell global routing problem, and describes the new LocusRoute approach to solving it.

2.1 Problem Definition

Global routing for standard cells decides the following for each wire in the circuit:

1. For each group of electrically equivalent pins (pin clusters) it determines which of those pins are actually to be connected.

2. If there is no path between channels when one is required, it must decide either which built-in feedthrough to use or where to insert a feedthrough cell.

3. It decides which parts of a channel to use for a wire, including the use of two distinct wires in the same channel if this is desirable.

4. It must determine the channel to use in routing from a pad into the core cells.

In this discussion of global routing there will be no differentiation between feedthrough cells and built-in feedthroughs - they are referred to jointly as *vertical hops*. The decision to insert a feedthrough cell or use a built-in feedthrough is deferred to a post-processing step. This does result in some inaccuracy in the track count, and is discussed further in Section 3.4.

The objective of a global router is to minimize the sum of the channel densities of all the channels (hereafter called the *total density*). It is important to note that the total density can be traded off with the number of vertical hops, so to compare the total density of two global routings fairly they should both use the same number of vertical hops.

2.2 The Basic LocusRoute Algorithm

In the LocusRoute algorithm, each wire sequentially goes through the following five steps:

1. **Segment Decomposition.** A multi-point wire is decomposed into a minimum spanning tree of two-point *segments*, using Kruskal's algorithm [Krus56]. This algorithm has running time $O(n^2)$ in the number of pin clusters. The effect of the sub-optimality of this decomposition is discussed in Section 3.2 below.

2. **Permutation Decomposition.** The segments are further decomposed, if necessary, into *permutations*, which are the set of possible routes between each pin in a pin cluster.

3. **Route Generation and Evaluation.** A low-cost path is found for each permutation by evaluating a subset of the two-bend routes between each pin pair. The definition of the *cost* of a wire is given below, in Section 2.2.2. The permutation with the best cost is selected as the route for that segment.

4. **Reconstruct.** This step joins all the segments back together, and assigns unique numbers to distinct segments of the same wire in each channel. This is so that a channel router can distinguish between two segments and will not inadvertently join them together.

5. **Record.** The presence of the newly routed wire is recorded so that later wires can take it into account.

In addition, LocusRoute uses the iterative technique described in [Nair87]. Briefly, this means that after the first time all wires are routed, each is sequentially ripped up and then re-routed. By routing each wire several times (typically four is sufficient), the final answer is improved by five to ten percent because later wires can take earlier wires into account after the first iteration. This also reduces the effect of the wire order dependency.

The details of the second, third and fifth steps above are described in the following sections. The others are simple enough that the above description suffices.

2.2.1 Decomposition into Permutations

Each two-point segment consists of pairs of *pin clusters* that contain electrically equivalent pins. The LocusRoute algorithm considers routes between every pin in one cluster and every pin in the other cluster. Each such route is called a *permutation*. Figure 1 illustrates three of the four possible permutations between clusters A and B, which have two pins each. The four possible permutations are: (A_1, B_1), (A_1, B_2), (A_2, B_1), (A_2, B_2). If clusters A and B are separated by only a short horizontal distance, then the (A_1, B_2) permutation is most likely the least-cost path of the four. If the horizontal distance is large then it is possible that any one of the four permutations could have the low-cost path, and

hence all should be investigated. This has been confirmed experimentally, and a constant horizontal separation (300 routing grids) has been determined beyond which total density will improve if all four permutations are evaluated.

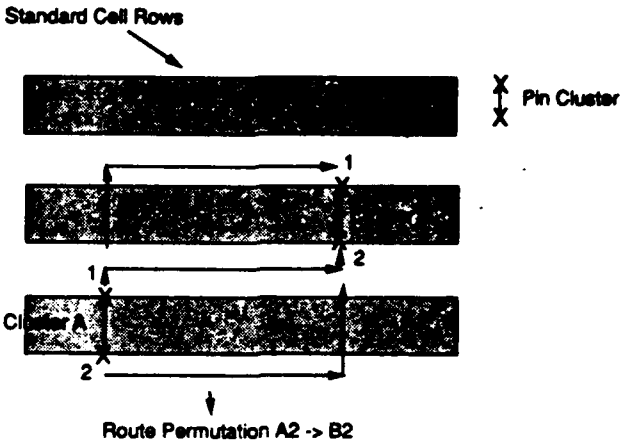

Figure 1 - *Permutation Decomposition of Segment*

2.2.2 Route Evaluation

The route evaluation step introduces two crucial notions of the LocusRoute algorithm: the cost model, which dictates the cost assigned to a path chosen for a wire, and the basic method of choosing routes based only on paths that have two or less bends.

Cost Model. Each possible routing position in a channel (also called *routing grid* of that channel) is represented as one element of an array as shown in Figure 2. The array, called the *Cost Array*, has a vertical dimension of the number of rows plus one, and a horizontal dimension of the width of the placement in routing grids. Each element of the Cost Array contains two values: H_{ij} and V_{ij}. H_{ij} contains the number of wire routes that pass horizontally through the grid at channel i in position j. This value changes as wires are routed. Similarly, V_{ij} is the cost, assigned by parameter, of traversing a row in travelling from channel i to channel $i + 1$ at grid position j. A wire is represented as a list of (i , j) pairs of locations in the Cost Array, corresponding to the locations of pins to be joined.

This model implies that more than one vertical hop can exist in one grid location, and that the assignment of a vertical hop does not disturb the placement. While these assumptions are strictly incorrect, their effect is minimal as discussed in Section 3.4.

Under this model, the objective is to find a minimum-cost path for each wire. The wire's cost is given by the sum of all of the H_{ij} and V_{ij} that it traverses. After a path is found for a wire that goes through location (i , j) its presence is recorded in the Cost Array (the appropriate H_{ij} and V_{ij} are incremented) so that subsequent wires can take it into account. The more wires going through a particular location in a channel, the less likely it is that area will be used. Note that in this model the total density is not directly minimized, but rather a combination of average density and wire length.

Two-Bend Route Generation and Evaluation. The LocusRoute algorithm searches for a low-cost path

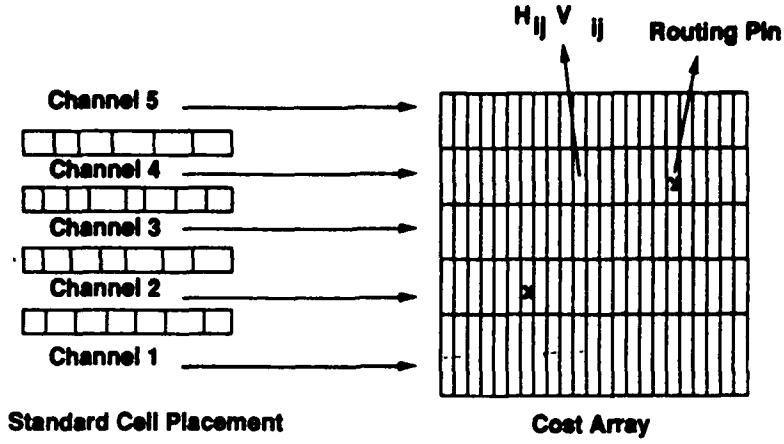

Figure 2 - *Cost Model*

for a permutation by *evaluating* the cost of a number of different routes and choosing the best. The basic approach is to evaluate a subset of all two-bend routes between the two pins, and then choose the one with the lowest cost. Generation of two-bend routes is discussed in [Ng86]. Figure 3 illustrates three possible two-bend (or less) routes inside a representation of the Cost Array as a small example.

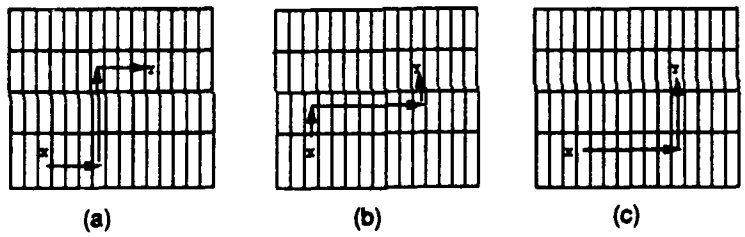

Figure 3 - *Sample Two-Bend Routes*

If the horizontal distance between the two pins is H routing grids, and the vertical difference is C channels then the total number of possible two-bend routes is $C+H$. In the LocusRoute algorithm the percentage of all the possible two-bend routes to be evaluated is a parameter. If fewer than 100% of all the routes are to be evaluated, the set of all possible routes is prioritized as follows: first all principally horizontal routes (those with bends only at the left and right extremes) are evaluated. Then the principally vertical routes (those with bends at the upper and lower extremes) are evaluated. Horizontal routes are evaluated first because it is important that all of the potential channels for the route be examined at least once. Within the horizontal and vertical groups, routes are searched in bisection order; i.e. if the limits of the group span are normalized to [0,1] then the routes are prioritized as $0, 1, \frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}$, and so on. This ensures that the possible space of routes is evenly spanned.

To calibrate the number of two-bend routes to be evaluated the two-bend router was compared against a least-cost path maze router. Both routers were not allowed to go beyond the bounding box of the two end points of the segment. Experimentally, it was determined that if only 20% of the two-bend routes were evaluated, then this would result in a path as good as that found by the maze router, as compared on the basis of total density for the entire circuit. On all of the test circuits used in the experiments discussed in the Section 3, the LocusRoute router's total density was within 2% of that obtained by the two-point maze router, with one exception of 3.3%. Most of the differences were below 1%. This is surprising in that the maze router looks for not only two-bend routes but for three or more bend routes. It implies that two-bend routes provide a sufficiently rich route set for the standard cell routing problem.

2.2.3 Recording A Wire

The last step in the algorithm is to record the presence of the wire's route in the Cost Array, so that the cost of using any part of that path will increase for other wires. This is done simply by incrementing the appropriate cells of the cost array. In the next iteration, the wire is "ripped up" by decrementing those same cells of the Cost Array.

3 Performance Comparisons

This section compares the quality and execution time of LocusRoute with three other routers.

3.1 Comparison with TimberWolf

Table 1 shows a comparison between the LocusRoute global router and the TimberWolf 4.2 [Sech85] global router for several industrial circuits. These circuits are from several sources: The standard cell benchmark suite (Primary1, Primary2, Test06 [Prea87]), Bell-Northern Research Ltd. (BNRA->BNRE), and the University of Toronto Microelectronic Development Centre (MDC). The placement for all of the circuits was done by the ALTOR standard cell placement program [Rose85, Rose88a]. Table 1 gives the number of wires in each circuit, the total density achieved by LocusRoute and Timberwolf, and the percentage fewer tracks LocusRoute achieved over Timberwolf. LocusRoute achieves significantly better total density than does the TimberWolf global router, ranging from 16% to 37% fewer tracks. The principal reason is that the TimberWolf global router is constrained to use only the minimum number of vertical hops, whereas LocusRoute uses considerably more. This is a reasonable practice in current technology because many standard cells contain "free" built-in feedthroughs. The execution times of LocusRoute and TimberWolf are comparable for most of the examples, though TimberWolf is faster by a factor of 8 and 3 respectively for circuits Test06 and Primary2. This is due to the fact that the LocusRoute algorithm increases in running time proportional to the area covered by the wire, which is much larger in these two circuits, and the inefficiency of the segment decomposition for large wires.

Circuit Name	# Wires	Total Density		
		LocusRoute	TimberWolf	% Fewer
BNRE	420	138	179	22%
MDC	575	150	179	16%
BNRD	774	188	225	16%
Primary1	904	262	316	17%
BNRC	937	202	247	18%
BNRB	1364	320	442	27%
BNRA	1634	315	432	27%
Test06	1673	335	537	37%
Primary2	3029	563	702	20%

Table 1 - *Comparison of LocusRoute and TimberWolf*

3.2 Comparison with Maze Router

For comparison purposes a maze router [Lee61] was developed, using the same cost model as LocusRoute, that exhaustively determines the optimal solution to the two-point routing problem. It also determines a good approximation to the minimum-cost Steiner tree for multi-point wires using the approach described in [Aker72]. The maze router was carefully optimized for speed. Table 2 shows the comparison of total density and execution time for the maze router and the LocusRoute router, for all of the test circuits. The comparison is made on the basis of roughly equal numbers of vertical hops. Execution times are for four iterations over all wires on a DEC Micro Vax II.

Circuit Name	Total Density			Time (Micro Vax II s)		
	Locus	Maze	Diff	Locus	Maze	Factor
BNRE	138	129	7%	88	2378	27x
MDC	150	141	6%	178	3173	18x
BNRD	188	182	3%	167	3306	20x
Primary1	262	255	3%	325	6534	20x
BNRC	202	189	7%	363	7250	20x
BNRB	320	308	4%	599	15116	25x
BNRA	315	294	7%	769	19652	26x
Test06	335	316	6%	5137	92272	18x
Primary2	563	549	3%	3758	48295	13x

Table 2 - *Comparison of LocusRoute and Maze Router*

For all circuits the LocusRoute total density (total number of routing tracks) is no greater than 7% more than that achieved by the maze router, and in some cases is as little as 3% more. Most of this difference is due to the sub-optimality of dividing the wires up into two point nets. LocusRoute ranges from 13 to 27 times faster than the maze router. Since the purpose of this work is to use the router as an area-based cost function for a placement algorithm, we will always be willing to trade this slight loss in quality for such a large gain in speed. This will allow many more potential placements to be evaluated.

3.3 Comparison with the UTMC Highland Router

For two of our circuits, we can also compare the total routing density with the United Technologies global router used in the recent benchmark effort at the 1987 Physical Design Workshop [Prea87,Robe87]. The placements used above for circuits Primary1 and Primary2 were also routed by the UTMC router. Table 3 shows the comparison of total density for both circuits, with each router using roughly the same number of vertical hops. The total density of the UTMC router for circuit Primary1 is notably less than for the LocusRoute router. This is probably due to the fact that the UTMC router also performs neighbour exchanges and cell orientation changes on the placement in order to reduce the total number of tracks. The LocusRoute total density for circuit Primary2 is slightly less than that achieved by the UTMC router. We have no information on the execution time of the UTMC router, except that for circuits near the size of Primary2, it would take roughly 10000 Vax 11/780 seconds [Robe87] which is about three times slower than LocusRoute.

Circuit Name	# Wires	Total Density	
		LocusRoute	Highland
Primary1	904	253	194
Primary2	3029	560	562

Table 3 - *Comparison of LocusRoute and UTMC Highland Router*

3.4 Effect of Vertical Hop Approximation

As discussed in Section 2.1, the abstraction of vertical hops (representing both feedthrough cells and built-in feedthroughs), and the fact that they *overlay* active cells, causes an inaccuracy in the track counts reported here. The difference is small, however. The 904-wire Primary1 circuit global routed to 249 tracks, using 995 vertical hops under the LocusRoute algorithm. The actual, post-process track count using 10 feedthrough cells and 985 built-ins was 253, only 1.6% more tracks. For the 3029-wire Primary2 circuit with 3424 vertical hops (287 feedthroughs, 3137 built-ins) the approximate track count was 546 and the post-process count was 590, an increase of 8%.

4 Parallel Decomposition and Implementation

As mentioned in the introduction, previous parallel routers have focused on fixed hardware implementations of the maze routing algorithm [Lee61]. A more flexible approach is to use general purpose parallel processors, which can be adapted to many applications. Using the flexibility of a general purpose multiprocessor, several "axes" of parallelism can be exploited. If these axes are *orthogonal* to

each other then when used in tandem they can achieve significant speedup. Two approaches to parallelizing an algorithm are said to be orthogonal if, when used together, the resulting speedup is the product of the speedup of the individual methods. In this section several ways of parallelizing the LocusRoute router are proposed and implemented:

1. **Wire-based Parallelism.** Each processor is given an entire multi-point wire to route.

2. **Segment-based Parallelism.** Each two-point segment produced by the minimum spanning tree decomposition is routed in parallel.

3. **Permutation-based Parallelism.** Each of the four possible permutations, as discussed in Section 2.2.1, are evaluated in parallel.

4. **Route-based Parallelism.** Each of the possible two-bend routes for every permutation are evaluated in parallel.

Note that these are only *potential* axes of parallelism. It is possible to eliminate some of them as uneconomical by using statistical run-time measurements of the serial router. For example, the number of two-point segments that actually need to have all four permutations evaluated is quite small with respect to the total. Thus, permutation-based parallelism is not going to provide significant speedup. Other measurements show that the time spent evaluating the cost of two-bend routes ranges from 50 to 90 percent of the total routing time and so reasonable speedup from route-based parallelism can be expected.

The following sections gives the details of three axes of parallelism, their performance and a quantitative measure of the degradation in quality if there is some. All decompositions assume a shared-memory multiprocessor.

4.1 Wire-Based Parallelism

In Wire-Based parallelism, each multi-point wire is given to a separate processor, which runs the LocusRoute routing algorithm as described in Section 2. The Cost Array is a shared data structure to which all processors have read and write access. This is an excellent axis of parallelism: if the sharing of the Cost Array does not cause performance degradation due to memory contention, and there are enough wires to provide good load balance, then the speedup should simply be the number of wires that are routed in parallel. The resulting parallel answer, however, will not necessarily be the same as the sequential answer. The problem is that the sequential router has complete knowledge of all wires that have already been routed, by virtue of their presence in the cost array. The parallel router has less information because it doesn't see the wires that are being routed simultaneously. The more wires routed in parallel, the less information each processor has to choose good routes that avoid congestion and hence cause an increase in total density. Thus the total density will increase as the number of processors increase. The measured effect on total density is discussed below, in Section 4.1.1.

4.1.1 Wire-Based Parallel Performance

Figure 4 is a plot of the speedup versus number of processors for the 3029-wire (Primary2) example running on an sixteen-processor Encore MULTIMAX. The speedup for p processors, S_p is calculated as $\frac{T_1}{T_p}$, where T_1 is the execution time on one processor and T_p is the execution time using p processors. The Encore uses National 32032 chip sets which, in our benchmarks, timed out slightly faster than a DEC Micro Vax II.

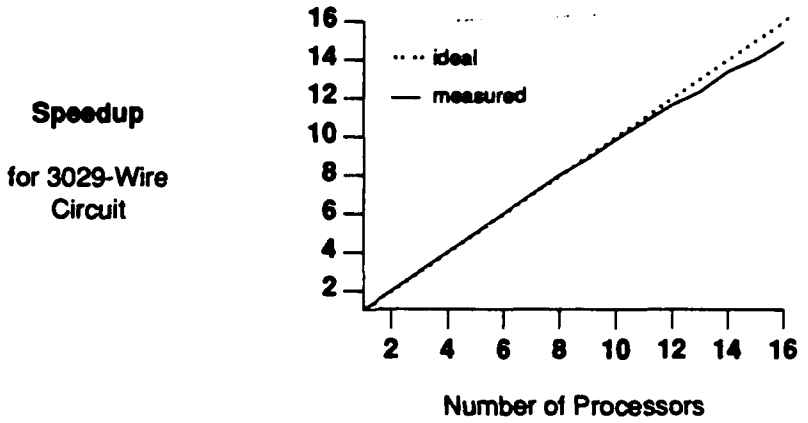

Figure 4 - *Wire-Based Speedup for Circuit Primary2*

It is clear from the figure that the wire-based approach achieves excellent speedup. Note that the execution time is only the actual routing computation time, excluding input time. For this circuit the increase in total density (between 1 and 16 processors) is negligible, and the number of vertical hops increases about 3%.

Table 4 gives the speedup using fifteen processors for the other test circuits. The speedup ranges from 10.1 for a smaller circuit to 14.1 for the largest. The speedup is less for smaller circuits because they are done in such a short time, so that the startup overhead and load balance become factors. The execution time is for four iterations over all the wires. It was discovered that very large global wires, such as TRUE or FALSE that have up to 150 pins, caused a severe degradation in speedup. This is because our system handles those nets just like any other, and the $O(n^2)$ nature of the minimum spanning tree algorithm causes load balancing problems. Since most production systems treat TRUE and FALSE signal nets differently (usually tapping directly into the power lines with special cells) these were eliminated under the assumption that they could be handled quickly that way.

Table 5 gives the density and vertical hop counts for both 1 and 15 processors using wire-based parallelism. The degradation in total density ranges between 1% to 7%. The increase in vertical hops is 6% or less. Again, in the context of using the router as a placement cost function, it is worthwhile to trade a small loss in quality for a large gain in speed, so that many more placements may be evaluated.

Circuit Name	1-Processor Time (s)	15-Processor Time (s)	15-Processor Speedup
BNRE	67	6.5	10.4
MDC	76.6	7.5	10.1
BNRD	136	11.8	11.5
Primary1	275	24.9	11.0
BNRC	196	16.9	11.6
BNRB	553	48.6	11.4
BNRA	713	54.9	13.0
Test06	5654	425	13.3
Primary2	3934	279	14.1

Table 4 - *Wire-Based Parallelism Speedup*

Circuit Name	Density			Vertical Hops		
	1-Proc	15-Proc	%Increase	1-Proc	15-Proc	%Increase
BNRE	130	137	5%	449	474	6%
MDC	134	142	6%	241	249	3%
BNRD	176	182	3%	530	574	6%
Primary1	262	271	3%	940	947	1%
BNRC	191	192	1%	725	739	2%
BNRB	307	328	7%	1904	1990	5%
BNRA	298	312	5%	2106	2198	4%
Test06	318	339	7%	3221	3309	3%
Primary2	560	593	6%	3053	3133	3%

Table 5 - *Wire-Based Parallelism Quality*

4.1.2 Gain Due to Removal of Locks

An interesting issue is whether or not each processor should lock the Cost Array as it both rips up and re-routes wires in the Cost Array. The act of ripping up a route is essentially a decrement, and re-routing is an increment on a set of cells in the Cost Array. Locking the Cost Array during these operations ensures that two simultaneous operations on the same element does not prevent one of the operations from being lost. It does, however, cause a significant performance degradation. For example, for the Primary1 circuit the speedup decreased from 8.3 to 6.4 using 15 processors when Cost Array locking was used. For the Primary2 circuit the speedup for 15 processors was reduced to 12.1 from 13.0 due to locking.

The final routing quality, however, does not decrease when locking is omitted. The reason for this is that the probability of two processors accessing the *same* Cost Array element (of which there are on the order of 10000) at the *same* instant is very low. Even if very few increment or decrement operations are lost, the effect on final quality is negligible since only a few elements would be wrong by a small amount. This was shown experimentally by performing ten runs with 15 processors on each of the above circuits, for both the locking and non-locking cases. For the two circuits table 6 gives the average running time, and the average and standard deviation of the total density and number of vertical hops. From this table it can be seen that the quality in both cases is very nearly the same. Note that in a placement context in which many more wires will be ripped up and re-routed, the effect of these small errors would be cumulative and so an occasional correction step may be necessary if locks are not used.

Circuit & Lock Type	Avg T (s)	Density		Vertical Hops	
		Avg.	SD	Avg	SD
Primary1 Locks	43.8	269	2.0	962	4.9
Primary1 NO Locks	33.7	272	3.0	964	3.4
Primary2 Locks	325	591	1.9	3126	7.5
Primary2 NO Locks	303	591	4.9	3122	4.0

Table 6 - *Speed & Quality Using and Not Using Locks*

4.2 Segment-Based Parallelism

In segment-based parallelism, each two-point segment of a wire is given to a different processor to route. This is the stage following the minimum spanning tree decomposition, but prior to the evaluation of different two-bend routes. Measurements of the sequential router showed that about 60% of the routing time was spent on wires with more than one segment. This means that a speedup of about two might be expected using three processors. Even though there are many wires that provide two or three-way parallel tasks, however, the size of those tasks are not necessarily equal. The amount of time taken by the LocusRoute router to route two points is proportional to the Manhattan distance between the two points. If, in a three-point wire, two of the points are close together and the third is far away, it will then take much longer to route one segment than the other. The processor assigned to the short segment will be idle while the longer one is being routed. This unequal load prevents a reasonable speedup. On the test circuits a speedup of about 1.1 using two processors was measured.

It is fairly clear, however, that an extra processor could be assigned to a *number* of processors that are routing different wires. It is likely that at any given time, one of them will be able to use the extra processor to route an extra segment. This technique would become essential in wire-based parallelism if the number of processors were increased much beyond sixteen. In that case, the load balance becomes a problem because wires with many segments take much longer than wires with few segments. Hence segment-based parallelism could be used as a method to balance those loads and speed up the routing of larger wires.

4.3 Route-Based Parallelism

In route-based parallelism all of the two-bend routes to be evaluated are divided among the processors. Each finds the lowest-cost path among the set of two-bend routes that it is assigned. When all processors finish, the route with the best overall cost is selected. In this case the processor loads are well balanced because the routes are all of the same length, and the number of routes is evenly divided among the processors.

Figure 5 is a plot of the speedup versus number of processors for the circuit Test06, a large circuit. It achieves a speedup of 4.6 using 8 processors.

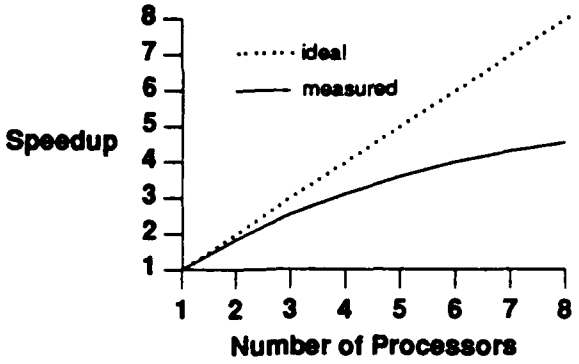

Figure 5 - *Route-Based Speedup for Test06*

Table 7 gives the best speedup achieved for all of the test circuits, ranging from 1.2 using 2 processors to 4.6 using 8 processors. The principal reason for the limitation in speedup is the sequential portion of the routing: the wire decomposition and the post-route processing that places the presence of the route into the Cost Array. On the small circuits that have lesser speedup, the sequential portion is about 50% of the total routing time, while on the larger circuits which have better speedup the sequential portion ranges from 10-15%. Another reason is that some segments have only one potential route, limiting the available parallelism.

5 Combining Two Orthogonal Axes of Parallelism

The wire and route axes of parallelism introduced above are orthogonal, and so when they are combined we can expect a multiplication of their respective speedups. In this section experiments are performed to demonstrate this effect on the Encore MULTIMAX. Using a simple model, the speedup for a larger number of processors is then predicted.

5.1 Implementation on the MULTIMAX

Because there are different kinds of tasks to be executed, the major challenge of combining the wire and route axes of parallelism is the *scheduling* of those tasks. An obvious static scheduling strategy is implied by the notion of orthogonality: for each wire that is being routed simultaneously by one processor in the wire-based approach, we now statically assign a constant number of processors to that wire to aid in the parallel execution of the route-based tasks. This situation is depicted in Figure 6. These

Circuit Name	Best Route-based Speedup (Speedup/#Processors)
BNRE	1.2/2
MDC	1.3/2
BNRD	1.3/2
Primary1	1.8/3
BNRC	1.6/3
BNRB	2.1/4
BNRA	1.9/4
Test06	4.6/8
Primary2	3.3/5

Table 7 - *Performance of Route-Based Parallelism*

extra processors are used only during the two-bend route evaluation.

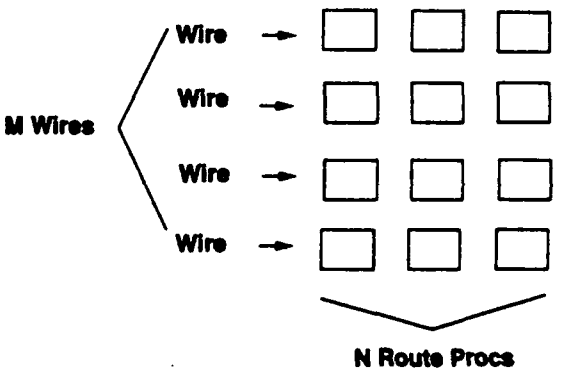

Figure 6 - *Static Scheduling Policy*

Several experiments were performed to show that the combined speedup of the wire and route-based approaches will indeed be the multiplication of the individually measured speedups. Table 8 gives the result of those experiments for the 3029-wire Primary2 circuit. For each experiment it gives the number of wires being routed in parallel (M), the number of processors assigned to each wire to do the routing tasks (N), the total number of processors ($M \times N$), the speedup predicted by multiplying the wire-based speedup using M processors and the route-based speedup using N processors, and the *measured* combined speedup. From this table it is clear that the speedups very nearly multiply, as expected. The small difference is due to increased contention for shared memory and the central bus, and the fact that two processors contend for one cache in the Encore MULTIMAX.

# Wire (M)	#Route Procs (N)	Total (M × N)	Speedup Predicted	Speedup Measured
3	4	12	9.0	8.7
4	3	12	9.9	9.6
6	2	12	10.8	10.3
3	5	15	10.4	9.9
5	3	15	12.4	11.7
7	2	14	12.6	12.0

Table 8 - *Static Schedule Experiments for Circuit Primary2*

A drawback of the static scheduling policy is that it cannot assign processors where they will be of best use. If one wire has very few routes while another has many, the processors assigned to the first are not used by the second. In addition, there is a portion of the wire routing procedure that only uses one processor, so the others will be idle. A dynamic scheduling approach allows any idle processor to be used by any wire that has a need for it. This was implemented as a single task queue. Wire processors add tasks to the queue, and other processors remove and execute tasks. The *granularity* of the routing tasks in the dynamic scheme, the number of two-bend routes assigned to one processor to evaluate per task, was tuned to achieve the best speedup. The best performance was achieved when the number of tasks was several times the number of available processors, indicating that the load balance effect was more significant than the overhead of starting up a task.

Experiments have shown that the dynamic approach can both obtain the same speedup as the static approach using fewer processors, or better speedup using the same number of processors. For example, for Primary2 the static approach attained a measured speedup of 9.9 using 15 processors, while the dynamic approach achieved 10.8.

5.2 Predicting Performance on More Processors

Since we have observed that the static schedule performance of the combined approach does indeed nearly multiply the speedups attained by the individual methods, it is possible to predict the performance of that schedule on many more processors. Assume, for a given circuit that a speedup of S_w is achieved using wire-based parallelism on W processors, and a speedup of S_r is achieved using route-based parallelism on R processors. Then, because the two approaches are orthogonal, the resulting speedup when they are used together should be $S_w \times S_r$ using $W \times R$ processors. This model neglects the effect of memory contention that may occur when the number of processors is increased dramatically. Table 9 shows the best predicted speedup for the test circuits. Combined speedup ranges from 13 using 30 processors to 61 using 120 processors. The smaller circuits are routed very quickly and so it is difficult to get speedups greater than 13 due to the startup overhead. The larger circuits benefit greatly from the combination of the approaches.

Circuit	$\frac{S_w}{W}$	$\frac{S_r}{R}$	$\frac{S_w \times S_r}{W \times R}$	A_1	A_{RW}
BNRE	$\frac{10.4}{15}$	$\frac{1.2}{2}$	$\frac{12.5}{30}$	46ms	3.7ms
MDC	$\frac{10.1}{15}$	$\frac{1.3}{2}$	$\frac{13.1}{30}$	38ms	2.9ms
BNRD	$\frac{11.5}{15}$	$\frac{1.3}{2}$	$\frac{15.0}{30}$	50ms	3.3ms
Primary1	$\frac{11.0}{15}$	$\frac{1.8}{3}$	$\frac{19.8}{45}$	89ms	4.5ms
BNRC	$\frac{11.6}{15}$	$\frac{1.6}{3}$	$\frac{18.6}{45}$	59ms	3.2ms
BNRB	$\frac{11.4}{15}$	$\frac{2.1}{4}$	$\frac{24.0}{60}$	127ms	5.3ms
BNRA	$\frac{13.0}{15}$	$\frac{1.9}{4}$	$\frac{24.7}{60}$	134ms	5.4ms
Test06	$\frac{13.3}{15}$	$\frac{4.6}{8}$	$\frac{61.2}{120}$	935ms	15.2ms
Primary2	$\frac{14.1}{16}$	$\frac{3.3}{5}$	$\frac{46.5}{80}$	358ms	7.7ms

Table 9 - *Predicted Combined Speedup of Wire and Route Parallelism*

Table 9 also contains the average routing time per net on one processor, A_1, and what the the average routing time per net would be under the maximum speedup, A_{RW}. That is, $A_{RW} = \frac{A_1}{S_w \times S_r}$. The average routing times for all circuits, under the various speedups range mostly from 3 to 6ms, (with one at 15ms) and approaches our goal of one to two milliseconds per net. If more processors were used under the wire-by-wire axis, this goal could definitely be achieved.

6 Conclusions

A new global routing algorithm for standard cells and its parallel implementation has been presented. The LocusRoute algorithm uses significantly fewer tracks than the TimberWolf standard cell global router, and is comparable to a maze router and an industrial router. It is more than a factor of 10 faster than either of the two latter routers. Three axes of orthogonal parallelism were developed to speed up the LocusRoute router further. Two of the three axes that were implemented achieved significant speedup - up to 14.1 using fifteen processors and 4.6 using eight processors. They should produce combined speedups of up to 61 times.

The Locus placement environment is currently being developed, and in the future will be combined with the parallel LocusRoute global router. Our aim is to achieve smaller final area by using the global routing as a better measure of each placement.

Acknowledgements

The author is grateful to John Hennessy for the encouragement and support of this work. Thanks to Tom Blank who provided many good suggestions for an earlier version of this paper. Thanks also to Grant Martin of Bell-Northern Research for the use of company circuits and to the people involved in the standard cell benchmark effort for supplying those test circuits. Carl Sechen provided version 4.2 of TimberWolfSC.

7 References

[Adsh82]
H.G. Adshead, "Employing a Distributed Array Processor in a Dedicated Gate Array Layout System," Proc. ICCC, September 1982, pp. 411-414.

[Aker72]
S. B. Akers, "Routing," Chapter 6 of **Design Automation of Digital Systems; Theory and Techniques**, M.A. Breuer, Ed., Englewood Cliffs, NJ, Prentice-Hall, 1972.

[Blan81]
T. Blank, M. Stefik, W. VanCleemput, "A Parallel Bit Map Processor Architecture FOR DA ALGORITHMS," Proc. 18th Design Automation Conference, June 1981, pp. 837-845.

[Breu77]
M.A. Breuer, "Min-Cut Placement," Journal of Design Automation and Fault-Tolerant Computing, Oct 1977, pp 343-362.

[Breu81]
M.A Breuer, K. Shamsa, "A Hardware Router," Journal of Digital Systems, Vol IV, Issue 4, 1981, pp. 393-408.

[Hana72]
M. Hanan, J.M. Kurtzberg, "Placement Techniques," Chapter 4 of **Design Automation of Digital Systems; Theory and Techniques**, M.A. Breuer, Ed., NJ, Prentice-Hall, 1972.

[Iosu86]
A. Iosupovici, "A Class of Array Architectures for Hardware Grid Routers," IEEE Transactions on CAD, Vol. CAD-5, No. 2, April 1986, pp. 245-255.

[Kamb85]
T. Kambe, T. Okada, T. Chiba, I. Nishioka, "A Global Routing Scheme for Polycell LSI," Proc. ISCAS 1985, pp. 187-190.

[Krus56]
J.B. Kruskal, "On The Shortest Spanning Subtree of a graph and the Traveling Salesman Problem," Proc. Amer. Math. Soc, 7, 1956, pp. 48-50.

[Lee61]
C.Y. Lee, "An Algorithm for Path Connections and Its Applications," IRE Transactions on Electronic Computers, Vol EC-10, pp 346-365, 1961.

[Lore88]
M.J. Lorenzetti, D. S. Baeder, "Routing", Chapter 5 of **Physical Design Automation of VLSI Systems**, B. Preas and M. Lorenzetti Ed., Menlo Park, Benjamin/Cummings Publishing, 1988.

[Nair82]
R. Nair, S.J. Hong, S. Liles, R. Villani, "Global Wiring on a Wire Routing Machine," Proc. 19th Design Automation Conference, June 1982, pp. 224-231.

[Nair87]
R. Nair,"A Simple Yet Effective Technique for Global Wiring," IEEE Transactions on Computer-Aided Design, Vol CAD-6, No. 2, March 1987, pp. 165-172.

[Ng86]
A P-C Ng, P. Raghavan, C.D. Thompson, "A Language for Describing Rectilinear Steiner Tree Configurations," Proc. 23rd Design Automation Conference, June 1986, pp. 659-662.

[Pate85]
A.M. Patel, N.L. Soong, R.K. Korn, "Hierarchical VLSI Routing - An Approximate Routing Procedure," IEEE

Transactions on Computer-Aided Design, Vol CAD-4, No. 2, April 1985, pp. 121-126.

[Prea87]

B.T. Preas, "Benchmarks for Cell-Based Layout Systems," Proc. 24rd Design Automation Conference, June 1987, pp. 319-320.

[Robe87]

Ken Roberts used the United Technologies Standard Cell global router on the standard cell benchmark placements. Results were discussed at the 1987 DAC.

[Rose85]

J.S. Rose, W.M. Snelgrove, Z.G. Vranesic, "ALTOR: An Automatic Standard Cell Layout Program," Proc. Canadian Conference on VLSI, November 1985, pp. 168-173.

[Rose88a]

J.S. Rose, W.M. Snelgrove, Z.G. Vranesic, "Parallel Standard Cell Placement Algorithms with Quality Equivalent to Simulated Annealing," IEEE Trans. on CAD, Vol. CAD-7, No. 3, March 1988, pp. 387-396.

[Rose88b]

J.S. Rose, "LocusRoute: A Parallel Global Router for Standard Cells," Proc. 25th Design Automation Conference, June 1988, pp. 189-195.

[Rute84]

R.A. Rutenbar, T.N. Mudge, D.E. Atkins, "A Class of Cellular Architectures to Support Physical Design Automation," IEEE Trans. on CAD, Vol. CAD-3, No. 4, October 1984, pp. 264-278.

[Sech85]

C. Sechen, A. Sangiovanni-Vincentelli, "The Timberwolf Placement and Routing Package," IEEE JSSC, Vol. SC-20, No. 2, April 1985, pp 510-522. pp. 432-439.

[Won87]

Y. Won, S. Sahni, Y. El-Ziq, "A Hardware Accelerator for Maze Routing," Proc. 24th Design Automation Conference, June 1987, pp. 800-806.

[Yama85]

M. Yamada, T. Hiwatashi, T, Mitsuhashi, K. Yoshida, "A Multi-Layer Router for Standard Cell LSIs," Proceedings ISCAS 1985, 191-194.